August, 1997

ASK ADVISOR

Visual FoxPro

Q: FoxPro provides a very powerful feature with the DateTime data type. When you subtract one DateTime value from another, the answer is the number of seconds between the two stored as a number. When it comes to displaying the result in a report, obviously the preferred format is hh:mm:ss. There appears to be no FoxPro function for this other than to write your own function using ABS() and INT(). Am I correct?

–Neville West (New Zealand via Internet)

A: You're right that there's no function to convert a number to a string in hh:mm:ss format. Fortunately, writing such a function is not hard. You're also right that the INT() function is useful here, but you don't need to use ABS() and do a lot of complex arithmetic. The combination of FoxPro's INT() and MOD() function makes the actual calculation portion of the function quite straightforward.

Here's SecToHMS which accepts a single numeric parameter and returns a string in the form "hh:mm:ss".

```
* SecToHMS.PRG
* This function converts a number to a string
* in the format HH:MM:SS.
* It accepts a single parameter - the number of seconds.
* If the parameter is not numeric or is negative
* or is greater than 359999 (the number of seconds in
* 99 hours, 59 minutes, 59 seconds - the most that fits
* in the format), it returns the empty string.
*
LPARAMETER nSeconds

#DEFINE SECONDS_PER_MINUTE 60
#DEFINE SECONDS_PER_HOUR 3600

ASSERT TYPE("nSeconds")="N" ;
       MESSAGE "Parameter must be numeric"

LOCAL cRetVal, nHours, nMinutes

IF TYPE("nSeconds")<>"N" OR nSeconds < 0 ;
   OR nSeconds > 99*SECONDS_PER_HOUR + ;
                 59*SECONDS_PER_MINUTE + 59
   cRetVal = ""
ELSE
   * Do the calculation
   nHours = INT(nSeconds/SECONDS_PER_HOUR)
   nSeconds = MOD(nSeconds,SECONDS_PER_HOUR)
   nMinutes = INT(nSeconds/SECONDS_PER_MINUTE)
   nSeconds = MOD(nSeconds,SECONDS_PER_MINUTE)
   cRetVal = PADL(nHours,2)+":" + ;
             PADL(nMinutes,2,"0") + ":" + ;
             PADL(nSeconds,2,"0")
ENDIF
```

```
RETURN cRetVal
```

There are several interesting things to talk about here. First, I use an assertion to test that the parameter is numeric. This is the kind of problem that usually should be able to be rooted out during testing. However, I test the range of the value at run-time because this function is likely to be called with calculated results that are not under the control of the developer. I also repeat the type test because assertions are probably off at runtime and with calculated values coming in, this belt-and-suspenders approach seems safer.

I define constants for the conversion factors involved. Often, #DEFINEd constants are used to allow changes to be made in a single location. In this case, the conversion factors are not likely to change (at least not for a few million years), but the constants make the code far more readable. Someone taking a quick look at the function can understand immediately why SECONDS_PER_HOUR and SECONDS_PER_MINUTE are used, where 3600 and 60 would require a moment's thought.

The final points are in the line that converts hours, minutes and seconds to a string. First, note that the PADL() function takes numbers directly and converts them to character - there's no need to use STR() and LTRIM(). Also, I chose to pad the hours portion with a space, but the minutes and seconds are padded with a zero, so that the output looks like " 1:01:01" rather than either "01:01:01" or " 1: 1: 1".

To use the function, just pass it a number. For example:

```
SecToHMS(30)
```

returns

```
" 0:00:30"
```

while

```
SecToHMS(4000)
```

returns

```
" 1:06:40"
```

and

```
SecToHMS(40000)
```

returns

```
"11:06:40"
```

This is the sort of function that I consider an extension of the programming language, so I'd put it in my common functions directory and have it included in any projects where it's appropriate.

–Tamar